atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
Tel: 512-615-7300
Fax: 512-615-7301
www.atsec.com

Sequoia Voting System v4.0
Source Code Review Team Report

September 19th, 2008

Table of contents

1 Executive Summary

This report was prepared by atsec information security corporation to review aspects of the security and integrity of the Sequoia Voting System v4.0. This document identifies the security vulnerabilities found through static code review that could be exploited to alter vote recording, vote results, critical election data such as audit logs, or to conduct a denial of service attack on the voting system.

A special emphasis was placed on re-examining the results of the UC Berkeley "top-to-bottom" review of a previous version of the source code. This document treats each of the findings identified in the UC Berkeley report and discusses the current state of these issues. Issues identified during the State of Washington review of the Sequoia product also were re-examined to verify that those issues are now resolved.

In addition, atsec performed top-to-bottom review of two new components, WinEDS Extended Services and WinEDS Election Reporting.

Finally, atsec reviewed the extent to which the system protects the integrity of ballot data or ballot images stored in the 400-C Central Count Scanner and Optech Insight® Plus.

Overall, we focused more on breadth than on depth. The source code was examined to the extent necessary to find specific examples of the relevant constructs or patterns. No attempt was made to exhaustively review the entire source code.

After examination of the source code, we have found that the security posture of the system is largely unchanged since the UC Berkeley review. There are still significant programming, logic, and architectural errors present in the software, ranging from fairly benign mistakes to severe systemic misuse of cryptographic methods and failures to implement correct integrity verification. The following are major points of weakness found in the source code:

- Failure to properly verify the integrity of election information, leaving the election definition and results vulnerable to undetectable tampering
- Improper and unsafe authentication and user account management
- Access control not implemented correctly allowing anyone with an account complete control over the central database
- Nonexistent or incorrect usage of encryption that can be easily defeated
- Lack of input sanitization leaving SQL injection vulnerabilities unchecked
- Various buffer overflow and software bugs creating possible avenues for arbitrary code execution inside the system

In some cases, specific previously-identified vulnerabilities have been mitigated, but evidence of similar vulnerabilities was found in the system, which suggests that a defensive software engineering approach is not yet fully implemented. The review team recognizes that such a change in the development process philosophy may take some time to become effective.

The review team also found that improvements had been made in addressing vulnerabilities highlighted in the UC Berkeley review. For example, passwords are no longer passed in plain text across the network. There were several cases where improvements had been made that, although not significantly affecting the overall security posture of the system in this version, will contribute to the security posture in future versions of the system once the system hardening is more mature.

Formal testing of the correct implementation of the AES algorithm contributes to the assurance afforded the system.

The methodology used for the atsec review was static source code review. Some findings reported in this document are subject to confirmation from functional tests and/or penetration tests.

2 Code Review Basis and Scope

atsec performed the code review on the basis of the Statement of Work, which states that the review shall place emphasis on security and integrity of the system and should identify any security vulnerabilities that could be exploited to alter vote recording, vote results, critical election data such as audit logs, or to conduct a "denial of service" attack on the voting system. The Statement of Work identified the following focus areas:

- Areas of special concern to be addressed include: Determining if the vulnerabilities found in the California Top to Bottom Review of the earlier version of the system have been addressed in this version of the system.
- Determining if a vulnerability discovered during certification testing by the State of Washington, where memory counters were not zeroed out and zero tapes did not reflect the contents of those counters has been addressed in this version of the system.
- Risks introduced in new modules of the system which have not existed in earlier versions on the system or which have not previously undergone a source code review for security.
- The extent to which the system protects the integrity of ballot data or ballot images stored on the 400-C Central Count Scanner and in the Optech Insight® Plus and which are used for computation of Rank Choice Voting results.

2.1 UC Berkeley report

The source code of an earlier version of the Sequoia Voting System was evaluated by a team at University of California, Berkeley (UCB) in July 2007 in a top-to-bottom review commissioned by the California Secretary of State. The report [UCB] identified two major types of problems in the Sequoia voting system:

- High-level systemic and architectural problems with implications for security
- Specific low-level software defects/issues

Based on these findings, the report provides a list of potential attacks on the system. atsec used this report as a basis for performing the code review of the system.

2.2 State of Washington certification testing

During certification testing of a previous version of the Sequoia Voting System v4.0 by the State of Washington in May 2008, two defects that adversely affect the accuracy of the vote count were discovered. These defects are referenced in the May 20, 2008 testimony by Ellen Theisen (VotersUnite!) [THEISEN]:

- The Insight optical scanner falsely reports that there are no votes recorded on the memory pack (the cartridge where votes are stored) when there actually are votes recorded on it. This is exactly the same as looking in a ballot box and claiming it's empty when, in fact, there are voted ballots in it.
- The WinEDS central tabulation system fails to recognize when there is a discrepancy between individual vote records and tabulated results stored on the memory pack, and it uses the individual vote records to calculate the results.

atsec used the information provided in the Theisen testimony as a basis for performing the code review of the system.

3 Code Review Methodology

3.1 Evaluation approach

As mentioned in section 2.1, a comprehensive evaluation of the Sequoia Voting System was completed in July 2008 by a team at UC Berkeley. This work provided invaluable information that was leveraged heavily by atsec in planning and executing the current source code review. In effect, leveraging the UC Berkeley work enabled the atsec team to focus on areas in which we could add particular value to the overall review effort: verifying progress on issues previously identified by the UC Berkeley team (and also several issues identified during State of Washington testing); closely examining the two modules not previously reviewed by other teams; and using NIST's Cryptographic Algorithm Validation System (CAVS) tool through the NIST Cryptographic Algorithm Validation Program-accredited Cryptographic and Security Testing (CST) laboratory (operated by atsec's CST) to test the correctness of the AES implementation that UC Berkeley had identified as worthy of greater attention. The atsec team did not explicitly perform line-by-line code review; rather, the team adopted the strategy of performing code review in the context of examination of the focus areas identified.

Therefore, in order to most effectively utilize the time and resources available, atsec focused its assessment on the following areas of interest:

1. evaluating code changes introduced in v4.0 of the system, including the two new modules (WinEDS Extended Services and WinEDS Election Reporting), to determine if the new software amounts to architectural changes addressing the high-level/systemic vulnerabilities identified in the UC Berkeley report.
2. evaluating code changes to determine if the specific defects outlined in Section 4 of the UC Berkeley report have been resolved:
    a. Issues 4.1.1 - 4.1.30 for WinEDS
    b. Issues 4.6.1 - 4.6.6 for Insight
    c. Issues 4.7.1 - 4.7.3 for MPR
    d. Issues 4.8.1 - 4.8.8 for Optech 400-C
3. evaluating code changes to determine if the issues identified during the State of Washington testing have been resolved.
4. top-to-bottom review of the two new modules (WinEDS Extended Services and WinEDS Election Reporting), including CAVS evaluation of the correctness of the AES algorithm implementation.
5. evaluating the integrity protection of ballot data and images for the 400-C Central Count Scanner and in the Optech Insight® Plus, which are used for computation of Rank Choice Voting results.

Due to the limited time available for the source code review, review of the two new modules focused more on breadth than on depth, meaning that a broad range of categories was examined based on the Statement of Work and the Voting System Standards of 2002. For each work item, the source code was only examined to the extent necessary to find specific examples of the relevant constructs or patterns. No attempt was made to exhaustively review the entire source code for the new modules.

The analysis contained in this report is based on the documentation and source code provided to atsec by the State of California. atsec made no attempt to validate that the source code we reviewed is the source code that was used to compile the Sequoia product as submitted to the State of California for testing.

The source code review was conducted manually and statically.

Some findings reported in this document are subject to confirmation from functional tests and/or penetration tests.

3.2 Threat model

During examination of the Sequoia Voting System v4.0, atsec adopted a threat model consistent with that of the UC Berkeley report. The model contains the following types of potential attackers:

Attacker type	Knowledge, access
Voter	Usually has low knowledge of the voting system machine design and configuration. Some might have more advanced knowledge. Might carry out attacks designed by others. They have access to the machine for less than one hour.
Poll worker*	Usually has a low knowledge of the voting machine design and configuration. Some might have more advanced knowledge. Might carry out attacks designed by others. They have access to the machine for less than one day
Election official insider	Has a wide range of knowledge of the voting machine design and configuration. They might have unrestricted access for long periods of time. Their designated activities include: - Setup and pre-election procedures - Election operation - Post election processing of results
Storage Personnel	Election or vendor employees that perform pre- and post-election maintenance and have access to the stored machines. Activities also include archiving and storage of the physical machines.
Vendor insider	Has a great knowledge of the voting system design and configuration. They have unlimited access to the machine before it is delivered to the purchaser and, thereafter, might have unrestricted access when performing warranty and maintenance service, and when providing election administration services,

3.3 Personnel

The source code review was performed by the following atsec information security corporation consultants:

Yi Mao, Jeremy Powell, Apostol Vassilev, Steve Weingart, Clemens Wittinger.

* Please refer to the note in section 3.7.1.1 regarding references to the Poll worker role in the context of the work papers.

3.4 Code review techniques

The techniques used by the team to perform the assessment are listed below.

Code security

- Evaluation of the likelihood of security failures being detected.
    - Are audit mechanisms reliable and tamper resistant?
    - Is data that might be subject to tampering properly validated and authenticated?
- Identification of trust levels, privileges/capabilities, and trust boundaries as appropriate for the scope
- Search for proper enforcement across trust boundaries
- Evaluation of whether the design and implementation follow sound, generally accepted engineering practices. Is code defensively written against:
    - bad data,
    - errors in other modules,
    - changes in environment,
    - user errors,
    - and other adverse conditions?
- Identify inputs and interfaces that appear susceptible in the intended use
- Include inputs and interfaces which are not intended for use in the assumed scope but which are available due to not being deactivated
- Verify if the code's assumptions are clearly identified in the design and descriptions
- Verify if the code includes checks appropriate for the assumptions
- Search for embedded, exploitable code (such as "Easter eggs") that can be triggered to affect the system
    - In addition to general observations as part of other work units, search specifically for constructs such as:
        - unexpected date/time dependent actions
        - control flow decisions based on specific input data, especially as related to non-admin users
        - administrative interfaces available during processing
- Search for dynamic memory access features which would permit the replacement of certificated executable code or control data or insertion of exploitable code or data
    - In addition to specific dynamic features, search for generic operating system or environment features that could be used for the same effect, such as:
        - user write access to any executable components including system startup and support
        - debuggers
        - tracing/instrumentation tools (strace)
        - dynamic library loading (LD_PRELOAD)

- Search for use of runtime scripts, instructions, or other control data that can affect the operation of security relevant functions or the integrity of the data.

Cryptography and key management

- Identify cryptographic functions based on design documentation and code examination
- Examine key generation, storage, transmission, use, and destruction
- Examine algorithms and their use compared to relevant standards, documentation, and best practices

Code quality and maintainability

- Check adherence to the Voting System Standards, 2002 and industry best coding practices
- Check for specific items from the Voting System Standards, 2002, such as use of numeric constants, use of control structures, control flow, assertions and debugging statements
- Check use of exceptions in relation to the general control flow
- Check block and inline comments for appropriateness, accuracy, and clarity, especially concerning assumptions made by the code, maintainer notes, high-level algorithm descriptions, and explanation of implementation choices
- Identify cases where interfaces or constructs are misused

3.5 Code review criteria

According to the Voting System Standards, 2002, Volume I, Section 4.2.6, the coding conventions used by the vendors should be one of:

- a vendor-identified published, reviewed and industry- accepted coding convention
- the coding conventions specified in VSS 2002, Volume II, Section 5

According to Sequoia coding standards, “As specified by paragraph 5.2.6 of the EAC standard: Volume I, SVS uses the following coding conventions:

a. The published, reviewed, and industry-accepted coding conventions used are used throughout the SVS applications

b. SVS code is written using the coding convention requirements specified in Volume II, Section 5.

c. Modules do not exceed 80 character line lengths, unless necessary for clarity and readability. No line length exceeds 120 characters.

d. Module line length excluding comments and header block should be less than 240 lines. Modules that exceed 240 lines of executable code when necessary for maintainability and clarity should have a Cyclomatic Complexity of less than 20. In addition, less than 10% of all modules should exceed 150 lines. Module line length should not exceed 350 lines of executable code regardless of Cyclomatic Complexity.

SVS encourages use of industry standard “best practices” coding techniques and performs regular code review sessions to assure adherence to these standards. However, when there is a compelling reason to divert from the standard, SVS documents the reason and the diversion.”

In addition the vendor specifies:

- PowerBuilder Coding Standards
- SQL Coding Standards
- C++ Coding Standards
- C# Coding Standards
- Visual Basic .NET Coding Standards
- Visual Basic 6 Coding Standards

atsec relied heavily on the analysis of the coding convention documented in the UC Berkley Report and did not perform a repeat analysis. However, reviewers were aware of the coding conventions specified and reported any consistent deviations. Particular focus was given to the two new modules, where the analysis, as already stated in section 3.1, was focused more on breadth than on depth, meaning that a broad range of categories was examined based on the Statement of Work and the Voting System Standards of 2002. For each work item, the source code was only examined to the extent necessary to find specific examples of the relevant constructs or patterns. No attempt was made to exhaustively review the entire source code for the new modules.

Summarizing, the analysis of the source code was more focused on finding serious coding bugs than on finding formal deviations from the relevant coding standards.

3.6 Standards and tools

3.6.1 Standards

- 2002 Voluntary Voting System Standards, Volume I, sections 4 and 7 and Volume II, section 5
- FIPS 180-2 with Change Notice 1, February 25, 2004
- Advanced Encryption Standard (AES), November 26, 2001
- FIPS 180-2, Secure Hash Standard (SHS), August 2002
- CERT C++ Secure Coding Standard (https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637)
- CERT C Secure Coding Standard (https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard)

3.6.2 Tools

These are the tools used for source code review.

Tool	Use	Source
Linux utilities – grep, vim, gcc, g++, python, make, ctags, diff	Overall code review	www.linux.org, www.python.org
VisualStudio 2005 Version 8.0.50727.42	Overall code review	www.microsoft.com
Cryptographic Algorithm Validation System (CAVS) v6.1 from NIST	Cryptographic algorithm implementation validation	http://csrc.nist.gov/index.html

Tool	Use	Source
AES CAVS test util, v1.0.3	CAVS testing for implementation of Advanced Encryption Standard (AES) in the WinEDS subsystem	atsec information security corporation

3.7 Supporting work papers and proprietary information

3.7.1 Work papers

Potential security issues discovered during testing were documented in work papers. The work papers contain proprietary information and are not included in the public report.

3.7.1.1 Poll worker role in the context of work papers

In some of our work papers, we include poll workers as potential attackers for the following reason:

Ten default roles are defined by the WinEDS access control system (please see section 4.1.1 in "Sequoia Voting Systems WinEDS System Operations Procedures" [WinEDS_SYSOP]). “Clerk” and “Tally Worker” are among the ten pre-defined roles. The responsibilities for the Clerk and Tally Worker roles, as for all other roles, are not specified in the vendor’s document. A small percentage of all poll workers might have enough knowledge of voting machines (including WinEDS work stations and WinEDS server) and sufficient skills to carry out an attack. Clerks and Tally Workers have accounts on a Sequoia Voting System, and so they must be considered to be potential attackers, capable of attacking some of the vulnerabilities identified in this report; for example, such an attacker could use his or her username and password appended by the database name to remotely access the WinEDS database at a later time through any SQL client tool without going through a WinEDS work station.

3.7.2 References to proprietary information

This report contains numerous markers ($*^1$, $*^2$, …$*^n$) that are references to source code or to information found in work papers or other documents containing proprietary information. Referenced work papers and documents are provided as an attachment to the private report, which also contains an index to map the references in this report to specific source code or documents in the attachment.

4 Evidence Received

4.1 Initial evidence

The atsec team received the source code and documentation files between 2008-08-13 and 2008-09-08 by email and by overnight FedEx delivery on separate CD-ROMs (see Evidence Received.pdf[*1]).

4.2 Late-arriving evidence

- Explanation of the "Pierce County, Washington error" correction to APX [PIERCE] (received on 2008-09-12)
- APX k2.16080408.1330 source code (received on 2008-09-12)

5 Assessment of Code Review Results

5.1 Assessment of UC Berkeley report high-level security architecture vulnerabilities

5.1.1 Data Integrity

(Reference the UC Berkeley report, section 3.1)

The atsec reviewer found that in the Sequoia system, there is a data input/output interface between the WinEDS component and the removable media MemoryPack and floppy disk. WinEDS is the Election Database System containing all the information about the election, including offices, ballots, voting machines, polling places, and results. MemoryPacks are used to carry ballot information and vote counts between WinEDS and the Insights, which are machines for scanning paper ballots, one by one, at a polling place. Removable media is used to carry ballot information and vote counts between WinEDS and the Optech 400-Cs. There is no effective mechanism to protect the integrity of data that is transferred via the removable media (see section 5.7).

The central component of WinEDS implements a client-server architecture. The client code does not sufficiently sanitize user input data before SQL statements are composed and executed on the Microsoft® SQL Server® (see section 5.6). There is a potential vulnerability for SQL injection attacks that result in unauthorized access to election data stored in the database or execution of malicious code on the database server machine to crash the system.

For every user, WinEDS creates an account with system administration privileges on the database server (see section 0). Users can access the database through a database client other than WinEDS. As a result, the access control on the database imposed by WinEDS can be bypassed. The integrity of data stored in the database is not protected. A user who directly accesses the database without going through the WinEDS user interface can add, delete and modify any data in the database.

5.1.2 Cryptography

(Reference the UC Berkeley report, section 3.2)

The atsec reviewer found that improper use of cryptographic methods throughout the Sequoia Voting Systems mentioned in the UC Berkeley report still remains an issue. Evidence of hardcoded encryption and hashing keys can still be found in several modules through the system[*2] (see section 5.3.2).

Also, the system makes use of cryptographic ciphers and hashes poorly. The SHA hash is still used as a Message Authentication Code (MAC) in a way known not to be secure. In the WinETP subsystem, the hardcoded 'hash key' is not even used in the hashing mechanisms. In practice, this means that there is no secret information used to create the MACs, rendering the authenticity of the message nonexistent, as replay attacks can be used to maliciously resend messages that have been overheard.

The CRC checksum algorithm, which is not cryptographically secure, is used in another MAC method. Although used by legacy code, the mechanism is still there for backwards compatibility. This provides an avenue of attack.

Many of the cryptographic facilities are implemented from scratch. For example, the reviewer found three separate SHA-1 implementations, one SHA-256 implementation, and one AES-128 implementation. It appears that the SHA implementations originated from the same file, but it was copied and pasted across the system when SHA was needed in a separate module, and then heavily edited to the point where the files are now very different.

Also, atsec has applied standard testing procedures to the WinEDS AES implementation.

5.1.3 Access control

(Reference the UC Berkeley report, section 3.3)

The atsec reviewer found that the architectural design of WinEDS access control has not been changed. It is based on user accounts and roles. Each user account has its own password and can be given one or more roles. Each role can be assigned a number of access rights that correspond to commands and buttons in the WinEDS user interface. A user with more than one role receives the combined access rights for each role. If a user lacks the right to access a particular command, the command will be disabled in the user interface.

WinEDS still maintains 111 components to which two types of access rights (i.e., Read and Update) can be assigned (see sections 5.2.16 and 5.2.17). The number of check boxes has been reduced by about 400, but the access control system remains complex. There is no easy way to manage access rights by grouping the components.

As was the case in the UC Berkeley testing, WinEDS continues to be a “thick client” (full-featured computer connected to a network) in which access control is provided by the application running on the client’s machine. Because the Sequoia design enforces access control only in the client, the entire access control system in WinEDS can be completely circumvented by communicating directly with the database.

Password management in WinEDS has been improved. Only the hash values of passwords are saved for verification during authentication. SHA-512 is the hash function, and it is provided by the Microsoft Crypto Library. However, due to an architecture defect, the strengthening of password management does not necessarily lead to a strengthened access control system.

5.1.4 Defensive Software Engineering

(Reference the UC Berkeley report, section 3.4)

Sequoia is a very complex software system that is developed in many different programming languages including assembly language, C, C++, C#, VB and PowerBuild. During the code review, the atsec reviewer identified the following issues that show the defensive software practices specified in the Voting System Standards 2002 [VSS] are not well carried out:

(1) User input data is not sufficiently sanitized[*3].

(2) Some of the user input data is not sanitized.

(3) Some modules contain thousands lines of code in one single file (see [VSS] vol 2, 5.4.2i).

(4) Array boundaries are not properly checked (see [VSS] vol 2, 5.4.2d).

(5) Type conversion that causes small buffer overflow is frequently used (see [VSS] vol 2, 5.4.2).

(6) Constants other than 0 and 1 are widely used (see [VSS] vol 2, 5.4.2u).

(7) Some dead code that is not referenced elsewhere and that seems to be left after revision from a previous version. This shows that the code is not well maintained and is not easy to review.

All these findings indicate that the coding practice is not completely compliant with the Voting Systems Standards 2002.

Note regarding conclusions in section 5.2-5.5

Issues that the reviewer was able to verify by source code review have been mitigated are marked **MITIGATED** or **PARTIALLY MITIGATED**. Issues that the reviewer found have not been addressed or that have been addressed but not mitigated are marked **NOT MITIGATED**. Issues that the reviewer could not verify by static source code review are marked **INCONCLUSIVE.**

5.2 Assessment of UC Berkeley report WinEDS security defects

5.2.1 WinEDS creates administrator-level database accounts for all users

Previous finding in the UC Berkeley report, issue 4.1.1:

For every WinEDS user account, WinEDS creates a corresponding account on the database server with administrator privileges, using the WinEDS account's username as the database account username and the WinEDS account's password, plus a short suffix, as the database account password. The suffix is the same for all accounts on all installations of WinEDS.

Current observations:

For every WinEDS user account, WinEDS creates a corresponding account on the database server with administrator privileges, using the WinEDS account's username appended with a profile database name as the database account username and the WinEDS account's password appended with the same profile database name as the database account password[*4].

Conclusion:

NOT MITIGATED Simply using a profile name instead of a short suffix while keeping the same design architecture is not sufficient to counter the previously identified vulnerability.

5.2.2 WinEDS does not encrypt or authenticate database communication

Previous finding in the UC Berkeley report, issue 4.1.2:

Database queries and responses are transmitted in the clear (i.e., without encryption) between WinEDS and the Microsoft SQL database. Also, communication with the database is not authenticated, so—if the database and the WinEDS client are running on different computers—it can be intercepted and altered in transit by anyone who has gained access to the local network.

Current observations:

The vendor's database installation guide (*cf.* [WinEDS_INSTALL], section 4.1.3, pp. 4-14) recommends the user to enable the option of using SSL for database communication.

The WinEDS source code does not show any support for the option of using SSL between WinEDS and the Microsoft SQL database. For instance, the function to make the database connection does not contain a flag parameter to indicate whether the connection should be encrypted or not.

Conclusion:

NOT MITIGATED No evidence was found in the WinEDS source code to indicate that authentication to the database and encryption of communication to and from the database is enforced.

5.2.3 WinEDS does not remove database access for deactivated users

Previous finding in the UC Berkeley report, issue 4.1.3:

After deactivating a user in WinEDS, the corresponding user account for the database is still active.

Current observations:

The WinEDS GUI does not have a command to delete user accounts. User accounts can only be deactivated in the Edit User dialog. This is confirmed by the vendors' document on system operation [WinEDS_SYSOP], *cf.* section 4.1.2.5, pp. 4-12.

There is no evidence to show that the function[*5] to drop a user from the database is called for deactivation of a WinEDS user. The reviewer did a global search on the entire set of the provided source code and found that only one procedure[*6] calls the function to remove a user from the database. This procedure validates that the username is a valid login, and then, based on the value of a flag parameter, either adds administrator privileges to this user or deletes this user from the administration group in two databases[*7]. The procedure is always created with the flag parameter set as "1" [*8], which means to add the administrator privileges to a given user. The reviewer did not find this procedure referenced elsewhere.

Conclusion:

INCONCLUSIVE Based on the source code analysis, the reviewer is not convinced that the developer has modified the code sufficiently to counter the previously identified vulnerability. However, this conclusion needs to be confirmed by functional testing to determine whether a deactivated user can still log into the database server via a database client.

5.2.4 WinEDS changes account usernames incorrectly

Previous finding in the UC Berkeley report, issue 4.1.4:

When the username of a user is changed in WinEDS, WinEDS changes its own record for the user but does not create a database account for the new username. This leaves the user unable to log in.

Current observations:

The reviewer did find any code changes in the user management component that could be responsible for fixing the identified issue.

Conclusion:

NOT MITIGATED The previously identified issue remains.

5.2.5 WinEDS does not encrypt password change requests

Previous finding in the UC Berkeley report, issue 4.1.5:

When a user changes her WinEDS password (as is required when she uses WinEDS for the first time, or by selecting Tools ! Change Password...), and the database is running on a different computer, the old and new passwords are passed in the clear over the network (with the password suffix attached) as parameters to a stored procedure in the database.

Current observations:

The hash value of username appended with a database name[*9] and the hash value of the password appended with the same database name[*10] are sent through the network to the database. The hash function used is SHA- 512 from Microsoft Crypto Library[*11].

Conclusion:

MITIGATED The passwords do not pass in clear text over the network. Their hash values travel through the network, instead. The previously identified vulnerability has been mitigated.

5.2.6 WinEDS retrieves the default password in the clear on every login

Previous finding in the UC Berkeley report, issue 4.1.6:

WinEDS assigns a default password to new user accounts. This default password is stored in the database and may be modified by an administrator. When a user logs in, WinEDS checks whether the user's password matches the default password; if so, the user is required to select a new password. In order to make this check, WinEDS retrieves the default password with a database request after every successful login.

Current observations:

The hash value of the default password is stored in the database, instead of the password itself. When a user logs in, the hash value of the default password is compared. The flag[*12] that indicates if the entered password matches the default password is used in place of the default password retrieved from the database[*13].

Conclusion:

MITIGATED WinEDS no longer retrieves the default password in the clear. The previously identified vulnerability has been mitigated.

5.2.7 WinEDS places the password suffix in a password entry field

Previous finding in the UC Berkeley report, issue 4.1.7:

When the user's password matches the default password, WinEDS presents a Password Expired window immediately upon login and forces the user to choose a new password. The Current Password field of this window contains the default password plus the password suffix (even though it is displayed as a string of asterisks).

Current observations:

In an initialization function*[14] for the Change Password window, the text box for displaying the old password is set to invisible for a new user or to reset a user's password[*15].

Conclusion:

MITIGATED The default password is not displayed in the Change Password window.

5.2.8 WinEDS displays the password suffix when resetting passwords

Previous finding in the UC Berkeley report, issue 4.1.8:

WinEDS provides a Reset Password feature that allows the WinEDS administrator to reset any user's password to the default password. The fixed password suffix is displayed in the WinEDS user interface whenever this feature is used.

Current observations:

The message displayed on the screen informs the user that the operation of the reset is successful. The message does not include a password. [*16].

Conclusion:

MITIGATED The default password is not displayed in the Reset Password window.

5.2.9 WinEDS changes the password for the administrator incorrectly

Previous finding in the UC Berkeley report issue 4.1.9:

For the account named sa (which is the default username for the system administrator account), WinEDS uses the entered password as the password for logging into the database, without adding the password suffix. However, the Change Password feature of WinEDS adds the password suffix to the newly chosen password.

Current observations:

The erroneous code identified in the UC Berkeley report has been corrected[*17]. WinEDS does not differentiate sa from other users. All users' passwords are appended with the profile database name before they are hashed.

Conclusion:

MITIGATED The modified code fixed the previously identified problem.

5.2.10 The WinEDS Data Wizard lets any user export data from the database

Previous finding in the UC Berkeley report, issue 4.1.10:

Using the Data Wizard tool in WinEDS, any WinEDS user can export almost any table in the Profile database. Any user with permission to open an election can export almost any table in the Election database as well. (The exceptions are tables containing binary columns, which the Data Wizard does not support.)

Current observations:

There is no evidence in the source files[*18] implementing the export utility to show that an access control mechanism is in place to check legitimacy for exporting the tables from the Profile or Election databases.

Conclusion:

INCONCLUSIVE The UC Berkeley report identified this problem through functional testing. The source code that is responsible for this problem is not identified. The reviewer checked export utility related source files and found no evidence regarding the access control that protects database tables. Functional testing is needed to determine if the problem is fixed.

5.2.11 The WinEDS Data Wizard lets any user erase any table in the database

Previous finding in the UC Berkeley report, issue 4.1.11:

Using the DataWizard tool in WinEDS, any WinEDS user can erase any table in the Profile database. Any WinEDS user that can open an election can erase any table in that election's Election database.

Current observations:

There is no evidence to show that the DataWizard imposes any access control mechanism on the table update operation on the Profile or Election database.

Conclusion:

INCONCLUSIVE The UC Berkeley report identified this problem through functional testing. The source code that is responsible for this problem is not identified. The reviewer checked the source code for updating a database table and found no evidence regarding the access control that protects database tables. Functional testing is needed to determine if the problem is fixed.

5.2.12 The WinEDS Data Wizard's import function does not work

Previous finding in the UC Berkeley report, issue 4.1.12:

The WinEDS DataWizard is documented and intended to be used for importing data into database tables, but this function does not work.

Current observations:

The variable that failed to place the table name in the SQL command and caused a SQL syntax error has been replaced by another variable[*19].

Conclusion:

MITIGATED The specifically identified code error has been corrected. However, the previously identified problem indicates that the developer did not conduct unit testing to catch this kind of coding error. Whether the import function works properly after correcting this specific one error is subject to functional testing. Moreover, if the import function works properly, then it may have impact on security. Functional tests and penetration tests for exporting and updating data using the WinEDS Data Wizard similar to those performed by the UC Berkeley team would need to be performed in order to determine whether the import function provided by the WinEDS Data Wizard lets any user import any data into the database.

5.2.13 List defect

Previous finding in the UC Berkeley report, issue 4.1.13:

WinEDS interprets a particular field stored in the database as a format string. The string can be entered in the WinEDS user interface or placed directly in the database. Four of the user roles supplied with WinEDS (Clerk, Administrator, PHASE I – Election Data, and PHASE II –Ballots) have sufficient permissions to enter the text string, but users in other roles may escalate their privileges to be able to enter the string as well.

Current observations:

The problematic code that was identified in the UC Berkeley report (reference to Annex: Item [20] in sequoia-source-annex.pdf) has not been changed[*20]. No data sanitization is done to validate the user input string.

Conclusion:

INCONCLUSIVE Based on the static code review, the previously identified problem seems to remain. However, this conclusion should be confirmed by functional testing.

5.2.14 WinEDS accepts negative vote totals from the database

Previous finding in the UC Berkeley report, issue 4.1.14:

When the official total for an election is calculated using Post Election ! Declare Winner, WinEDS does not check whether any vote totals stored in the database are negative. Any negative vote totals are included in the canvass report.

Current observations:

There is no evidence found in the related source files[*21,*22,*23] that checks whether any vote totals stored in the database are negative.

Conclusion:

INCONCLUSIVE Based on the static code review, the previously identified problem seems to remain. However, this conclusion should be confirmed by functional testing.

5.2.15 Some WinEDS default user roles allow users to gain administrative privileges.

Previous finding in the UC Berkeley report, issue 4.1.15:

WinEDS comes with ten default user roles. Besides users in the Administrator role, who have unlimited access to WinEDS, users with one of the roles PHASE I – Election Data, PHASE II – Ballots, PHASE III – Machine Programming, PHASE IV – Tally, PHASE V – Post Election, and PHASE VI – Archived have permission to assign roles to users, and can thus circumvent the WinEDS access control by assigning themselves the Administrator role.

Current observations:

WinEDS still defines the default user roles. The access privileges of the identified six default roles have been adjusted in a source file[*24] that allows them to have access to the expected components only.

Conclusion:

INCONCLUSIVE The security component in which an authorized role can assign roles to users is not accessible by the six default user roles previously identified in the UC Berkeley report. Based on the static source code review, the problem seems to have been fixed. However, this conclusion should be confirmed by functional testing.

5.2.16 WinEDS access rights are poorly described and documented

Previous finding in the UC Berkeley report, issue 4.1.16:

The WinEDS documentation describes user access capabilities in insufficient detail and overly broad terms.

Current observations:

As described in [WinEDS_SYSOP],the types of access have been reduced from six types (New, Edit, Remove, Query, Admin and All) to two types (i.e., Read and Update).

There are still 111 components in total[*25]. For each component, there are two types of access to assign. This still results in quite a big number of checkboxes to maintain.

Conclusion:

PARTIALLY MITIGATED If the system to manage Read and Update access rights were easily understandable, the lack of detailed user documentation would not be problematic. However, the WinEDS access control system remains difficult to understand and manage. There is still a potential that the complexity of the access control system could lead to configuration mistakes.

5.2.17 WinEDS access rights are difficult to minimize

Previous finding in the UC Berkeley report, issue 4.1.17:

The user interface for assigning access rights in WinEDS consists of 111 rows of checkboxes, with up to six checkboxes in each row. An administrator attempting to configure access rights must individually turn on or off these checkboxes. In particular, there is no easy way to clear all the checkboxes or manipulate them in groups.

Current observations:

There are still 111 components to which access rights Read, Update or both are assigned. There is no code change in the function[*26] that creates a new role. There is still no easy way to clear all the checkboxes or manipulate them in groups.

Conclusion:

PARTIALLY MITIGATED The number of check boxes has been reduced. As the design architecture has not been changed, the previously identified problem still exists on a smaller scale.

5.2.18 WinEDS fails to check some function return codes

Previous finding in the UC Berkeley report, issue 4.1.18:

We found several examples of code where WinEDS does not check the return code of a function for errors.

Current observations:

The return values of the functions[*27] that are identified in the UC Berkeley report still are not checked.

Conclusion:

NOT MITIGATED The previously identified problem remains.

5.2.19 WinEDS contains many small buffer overflows

Previous finding in the UC Berkeley report, issue 4.1.19:

On many occasions, WinEDS performs unchecked memory accesses which could potentially violate memory safety.

Current observations:

The two-byte overflow problem in the function[*28] is not fixed.

Conclusion:

NOT MITIGATED The reviewer did not attempt to confirm all of the overflow instances identified by the UC Berkeley report. The result of a spot check on the overflow issue indicates that the previously identified buffer overflow vulnerability still exists.

5.2.20 Integrity checking of Results Cartridges by WinEDS is not adequate to detect tampering

Previous finding in the UC Berkeley report, issue 4.1.20:

WinEDS does perform a check to detect whether the contents of a Results Cartridge have been tampered with. When writing to the Results Cartridge, the Edge adds a cryptographic message authentication code. However, apparently to provide backward compatibility, it is possible to construct a code that is accepted by WinEDS without access to any key material or other secrets

Current observations:

The function[*29] that validates a Result Cartridge always reports valid if the firmware version is lower than 4.0.

Conclusion:

NOT MITIGATED The previously identified problem remains.

5.2.21 WinEDS fails to check the integrity of election results.

Previous finding in the UC Berkeley report, issue 4.1.21:

WinEDS does not always check whether a cartridge has been tampered with. If certain checksums on a Results Cartridge are zero, or the Edge firmware version on an Audit Trail Cartridge is too small, or the cartridge is an Early Voting Cartridge, the checks are omitted.

Current observations:

The function[*30] that validates a Result Cartridge always reports valid if the firmware version is lower than 4.0.

Conclusion:

NOT MITIGATED WinEDS still does not always check whether a cartridge has been tampered with. The additional check condition is not sufficient to counter the previously identified vulnerability.

5.2.22 WinEDS does not prevent an integer overflow during preferential vote tallying

Previous finding in the UC Berkeley report, issue 4.1.22:

When WinEDS computes tallies for preferential contests from votes read from a Results Cartridge, malicious data on the cartridge can cause an integer overflow in WinEDS, which can be used by an attacker to write to memory addresses of her choice. We have no evidence that the code containing the weakness is ever executed.

Current observations:

The source file[*31] that contains the code for tallying preferential contests is no longer part of the system. As a result, the code for a buffer allocation (referenced in Annex: prefTable in line 82 of WinEDS/DoPrefTally.cpp referred by the UC Berkeley report) and the function (reference to Annex: CPreferentialTally::GetMaxPrefCount() in the UC Berkeley report) that reads a file on the Results Cartridge do not exist any more.

Conclusion:

MITIGATED The previously identified problem has been removed.

5.2.23 WinEDS trusts the list of precincts for which a Results Cartridge claims to report votes

Previous finding in the UC Berkeley report, issue 4.1.23:

A Results Cartridge can report results for an unlimited number of precincts other than the one it was assigned. If WinEDS tallies such a cartridge, it will accept all the votes on the cartridge, including for other precincts. These votes are included in the Statement of Vote and in some, but not all, reports generated by WinEDS.

Current observations:

Each vote reported on a Results Cartridge contains a Selection Code[*32] linking the vote to the precinct in which it was cast. WinEDS uses this information to determine in which precinct tally the vote on the Results Cartridge will be included. The logic and structure to identify the linkage between a Selection Code and precinct has not been changed.

Conclusion:

INCONCLUSIVE Based on the static code review, the previously identified problem seems to remain. However, functional testing is necessary to determine what reports are affected by this defect.

5.2.24 WinEDS writes to an array index read from the Results Cartridge without checking whether it is negative

Previous finding in the UC Berkeley report, issue 4.1.24:

The WinEDS function for printing results reports for a specific cartridge writes to memory using an array index specified by a file on the Results Cartridge, without checking whether it is negative. The file contains logical descriptions of each candidate.

Current observations:

WinEDS still does not check if an array index read from the Results Cartridge is negative. All the referenced code in the UC Berkeley report remains unchanged (reference to Annex: lines 154 and 213 in WinEDS_4.0.116_source\Image_080808_1128\Source\WinEdsCpp\source\AvcEdge\Reports\Endrs.cpp).

Conclusion:

NOT MITIGATED The previously identified problem is not fixed.

5.2.25 WinEDS increments integers in an array using unchecked array indices read from a Results Cartridge

Previous finding in the UC Berkeley report, issue 4.1.25:

For each candidate running in a race, the tallying algorithm for undervotes uses an entry from a file on the Results Cartridge as an index into an array on the heap, without checking that the entry actually corresponds to a valid index.

Current observations:

The tallying algorithm for undervotes still uses an entry from a file on the Results Cartridge as an index into an array on the heap[*33]. The entry still is not checked for its validity. Furthermore, the size of the array is still determined by the size of a file[*34] on the Results Cartridge. The memory location specified by the array pointer and the index still is incremented[*35] as a 32-bit long integer.

Conclusion:

NOT MITIGATED The code that causes the previously identified potential vulnerability has not been changed.

5.2.26 Some WinEDS reporting functions will never terminate if given a malformed Results Cartridge

Previous finding in the UC Berkeley report, issue 4.1.26:

A circular linked list read from a file on a Results Cartridge can cause a report generation in WinEDS to run forever. The linked list describes party endorsements for candidates.

Current observations:

A circularly-linked list is still attempted to be read from a file on a Results Cartridge using an array index[*36]. The termination condition for the loop is that the first element processed is encountered again[*37] to determine whether the list is circular.

Conclusion:

NOT MITIGATED The code that causes the previously identified problem has not been changed. WinEDS will still run into an infinite loop if the file to be read from the Results Cartridge is modified so that the list contains a loop excluding the first element.

5.2.27 WinEDS assumes the MPR is authentic and reliable.

Previous finding in the UC Berkeley report, issue 4.1.27:

WinEDS relies on the MPR to read and write election information on MemoryPacks, but it does not verify that the MPR is working correctly.

Current observations:

No code was found in the WinEDS module communicating with MPR[*38] that tests the functioning of the MPR or challenges the MPR to prove that it is an authentic MPR. The MPR only asks the MPR what software version it is running[*39] to start a conversation with MPR.

Conclusion:

NOT MITIGATED The code that causes the previously identified problem has not been changed. The vulnerability still exists.

5.2.28 WinEDS assumes that MemoryPack data received from an MPR is correct

Previous finding in the UC Berkeley report, issue 4.1.28:

WinEDS performs no checking to detect inadvertent corruption or tampering of vote counts received from the MPR. Vote counts are also not checked to determine whether they are in a reasonable range.

Current observations:

When the WinEDS receives vote counts from the MPR, it only checks a CRC value[*40]. No other verification code such as hash or signature verification on the vote counts was found in the WinEDS module that receives vote counts from MPR[*41].

Conclusion:

NOT MITIGATED WinEDS only performs CRC checking on the vote counts on the MemoryPack. CRC is too weak to guarantee the integrity of vote counts. The results received from the MPR are vulnerable to undetected tampering and, hence, are not trustworthy.

5.2.29 WinEDS assumes that the MemoryPack serial number received from the MPR is correct

Previous finding in the UC Berkeley report, issue 4.1.29:

The WinEDS database records which MemoryPacks are assigned to which precincts. When WinEDS receives election results from a MemoryPack, it looks up the MemoryPack by its serial number to determine which precinct those results are for. WinEDS obtains this serial number by asking the MPR to read a specific memory location on the MemoryPack. If the serial number corresponds to any known MemoryPack that has not already been processed, it is accepted. The serial number is a seven-digit number entered manually into WinEDS by the operator for individual Insights or generated sequentially by WinEDS for a series of Insights.

Current observations:

The WinEDS still reads the serial number of a MemoryPack[*42] to determine which precinct the results retrieved from the MemoryPack are for.

A method[*43] was added to verify the election information for a given serial number[*44]. This method is called[*45] before the vote results retrieved from the MemoryPack are tallied. However, the added method does not validate the serial number itself.

Another method[*46] was also added to verify whether the retrieved serial number together with the tally type ID matches the machine assignment information kept in WinEDS database[*47]. This method is called[*48] before the vote results retrieved from the MemoryPack are tallied.

Conclusion:

INCONCLUSIVE The modified code is not sufficient to address the root cause of the previously identified problem which is about the validation of the serial number. This conclusion needs to be confirmed by functional testing and penetration testing.

5.2.30 WinEDS does not notify the operator if a MemoryPack contains results for extraneous precincts

Previous finding in the UC Berkeley report, issue 4.1.30:

A single MemoryPack can contain results for many precincts. The WinEDS database records which MemoryPacks are assigned to which precincts. When WinEDS finds results on a MemoryPack for precincts

that don't correspond to the MemoryPack's serial number, it silently ignores them and proceeds without notifying the user.

Current observations:

There is no error message found in the WinEDS module[*49] that informs the user about extraneous precincts contained on a MemoryPack.

Conclusion:

INCONCLUSIVE The previously identified vulnerability seems to remain, but functional testing and penetration testing are necessary to confirm this conclusion.

5.3 Assessment of UC Berkeley report 400-C Central Count Scanner security defects

5.3.1 WinETP in counting station mode does not encrypt or authenticate its communication with master WinETP system

Previous finding in the UC Berkeley report, issue 4.8.1:

Communication with the master WinETP system is not encrypted or authenticated, so it can be intercepted and altered in transit by anyone who has gained access to the local network.

Current observations:

The two classes referenced by the UC Berkeley report[*50] that contain mechanisms for transferring election coding files have no revisions since after the UC Berkeley report was published. Furthermore, we cannot find any evidence elsewhere indicating that encryption or authentication is being applied.

Conclusion:

NOT MITIGATED An attacker with access to the local network can still conceivably intercept and change the election coding files without any detection.

5.3.2 WinETP contains a hardcoded key in the source code

Previous finding in the UC Berkeley report, issue 4.8.2:

There is a 16-byte key hardcoded as a constant in the 400-C source code and labeled as a "hash key."

Current observations:

This key can still be found in the source code[*51.]

Conclusion:

NOT MITIGATED Hardcoded keys are difficult to keep secret, and therefore an attacker with access to one 400-C system could still compromise every other 400-C system.

5.3.3 WinETP's hashing routine does not use the key passed in

Previous finding in the UC Berkeley report, issue 4.8.3:

The hashing module in the 400-C software takes a key as a parameter (such as one of the above hardcoded keys), but does not use the key.

Current observations:

The hashing routine[*52] still does not use the hash key. The revision comments also reflect that no changes have been made.

Conclusion:

NOT MITIGATED Using a secret hash key makes it very difficult to perform certain kinds of man-in-the-middle and replay attacks. Without this key, this functionality is not there. Also, the interface to this hashing routine is very misleading and could lead to incorrect assumptions about the quality of the hashes being produced.

5.3.4 WinETP stores the hash for the election coding file in the file itself

Previous finding in the UC Berkeley report, issue 4.8.4:

Optech 400-C Security Specification states that the election coding file, which describes the ballots, offices, precincts, and reports, contains a "hash of the file to prevent tampering" (page 2-1). This hash, which uses no key as mentioned above, is stored in the file itself.

Current observations:

The code writing the election header[*53] still saves the hash of the election coding (EC) file in the file itself. The revision comments also do no reflect new changes to the code since the UC Berkeley report.

Conclusion:

NOT MITIGATED Anyone wishing to change the EC file must only change the file, reapply the hashing method, and then write it back to the file.

5.3.5 WinETP does not check the integrity of the election coding file when the file is loaded over the network

Previous finding in the UC Berkeley report, issue 4.8.5:

Although WinETP checks the hash in the election coding file when it is loaded from disk, it neglects to perform this check on an election coding file loaded over the network.

Current observations:

The absence of integrity checks mentioned in the UC Berkeley report still exists. Following the code paths that load election headers over a network connection have no calls to hashing mechanisms[*54].

Conclusion:

NOT MITIGATED Combined with the fact that WinETP still does not implement an encrypted or integrity checked network connection, election headers can be modified through man-in-the-middle attacks without the system's knowledge.

5.3.6 Integrity checking by WinETP of the precinct results file is not adequate to detect tampering

Previous finding in the UC Berkeley report, issue 4.8.6:

WinETP uses a CRC to check the integrity of precinct results files. CRCs are not secure to detect intentional tampering.

Current observations:

The code still computes a CRC checksum[*55] instead of a cryptographic hash function.

Conclusion:

NOT MITIGATED As the UC Berkeley report mentions, anyone with write access to the floppy disk or storage media in which precinct results are stored can edit the results in such a way that WinETP will not detect the changes, because CRC is an insecure hashing algorithm.

5.3.7 The R-Code interpreter reads from an array without checking the array index, which is dictated by R-Code instructions

Previous finding in the UC Berkeley report, issue 4.8.7:

An R-Code program can cause the R-Code interpreter in WinETP to read data beyond the allocated size of an array.

Current observations:

The reviewer still believes this to be the case. There is no bounds checks on user input (in this case, a malformed election header size)[*56].

Conclusion:

NOT MITIGATED There is still a possibility of a buffer overrun in the code, with malformed election headers that might result in arbitrary code execution on the WinETP system.

5.3.8 The E-Code interpreter writes to an array without checking the array index, which is dictated by E-Code instructions.

Previous finding in the UC Berkeley report, issue 4.8.8:

An E-Code program can cause the E-Code interpreter in WinETP to write arbitrary data beyond the allocated size of an array, thereby controlling a region of memory 16 384 bytes long on the heap starting at the beginning of the array. The interpreter executes an E-Code program from the election coding file when a ballot is scanned.

Current observations:

As in the UC Berkeley report, the boundaries are not checked. However, a casual reviewer of the code may believe that Sequoia has fixed the code. The comments documenting the function mention that the function has bounds checking[*57], which is actually right. The issues is subtle in that the bounds against which it checks the counter is calculated in a different location[*58] depending on the election header that may not be well formed.

Conclusion:

INCONCLUSIVE The reviewer believes this to still be an issue after static analysis of the source code, but verification is needed through a live-code debugging.

5.4 Assessment of UC Berkeley report Optech Insight® Plus and MPR security defects

5.4.1 The HPX chip is physically replaceable

Previous finding in the UC Berkeley report, issue 4.6.1:

The HPX firmware controlling the behavior of an Insight resides on a removable and re-writable memory chip.

Current observations:

A photograph of the production version of the card shows the EPROM in a socket. The chip is removable.

Conclusion:

NOT MITIGATED The vulnerability remains.

5.4.2 APX on MemoryPack has full control of the Insight

Previous finding in the UC Berkeley report, issue 4.6.2:

The APX software loaded from the MemoryPack can execute arbitrary code and thereby have full command of the Insight.

See note following section 5.4.6

5.4.3 Integrity checking of the HPX and APX is performed by the HPX and APX themselves, respectively

Previous finding in the UC Berkeley report, issue 4.6.3:

The HPX checks its own integrity using a CRC. After it transfers control to the APX, the APX also checks its own integrity using a CRC. In terms of security against tampering, both of these checks are similar in nature to asking patrons at a bar to check their own IDs.

See note following section 5.4.6

5.4.4 Integrity checking of the HPX is not adequate to detect tampering

Previous finding in the UC Berkeley report, issue 4.6.4:

The integrity of the HPX is checked by the HPX using a CRC stored on the same memory chip in the Insight that contains the HPX. CRCs do not provide protection against intentional tampering.

See note following section 5.4.6

5.4.5 Integrity checking of the APX is not adequate to detect tampering

Previous finding in the UC Berkeley report, issue 4.6.5:

The integrity of the APX is checked using a CRC stored on the same MemoryPack that contains the APX. CRCs do not provide protection against intentional tampering.

See note following section 5.4.6

5.4.6 Integrity checking of the election parameters is not adequate to detect tampering

Previous finding in the UC Berkeley report, issue 4.6.6:

The integrity of the election parameters is checked using a CRC checksum stored on the same MemoryPack that contains with the election parameters. CRCs are not computed using a secret key.

Note regarding atsec observations for UC Berkeley report issues 4.6.2 – 4.6.6 (above) and 4.7.3 (below)

atsec performed a static source code review and confirmed the assertions that only a simple CRC is used to verify the integrity of the data and programming; no other mechanisms were found to protect the integrity of either the data or the APX programming in the MemoryPack or the HPX programming in the Insight Plus. This is also true of the MPR. If the attacker has access to an EPROM/FLASH burner, or an MPR (in the case of MemoryPack programming and/or data), either could be used to facilitate an attack.

Conclusion:
NOT MITIGATED The programming and data on the MemoryPack Rader and the Insight Plus®, both on the CPU card and in the Memory Pack does not have any real protection against tampering by a moderately skilled attacker.

5.4.7 The MPR does not have a physical locking mechanism

Previous finding in the UC Berkeley report, issue 4.7.1:

On the MPR provided to us by Sequoia, there was no mechanism for locking the case to prevent tampering. Removing four screws is sufficient to open the case.

Current observations:

The hardware has not changed.

Conclusion:

NOT MITIGATED The vulnerability remains.

5.4.8 The MPR's permanent software resides on a socketed EPROM

Previous finding in the UC Berkeley report, issue 4.7.2:

The permanent part of the MPR software resides on an EPROM (erasable, programmable memory chip) in a socket on the MPR main board.

Current observations:

A photograph of the production version of the card shows the EPROM in a socket. The chip is removable.

Conclusion:

NOT MITIGATED The vulnerability remains.

5.4.9 Integrity checking of MemoryPack results by the MPR is not adequate to detect tampering

Previous finding in the UC Berkeley report, issue 4.73:

The MPR checks the integrity of the vote counts on the MemoryPack using a CRC. CRCs do not provide protection against intentional tampering.

See note following section 5.4.6

5.5 Assessment of State of Washington testing defects

atsec performed a static source code review of the APX and HPX assembly language code for the Insight Plus with respect to the issues identified by the State of Washington testing as noted in the Theisen Testimony [THEISEN]:

- Incorrect vote reports from the Insight optical scanner
- Problems with recognizing discrepancies between individual vote records and tabulated results stored on the memory pack

As noted in the Pierce Error Correction [PIERCE], the condition that led to the error described in [THEISEN] was caused by a zero or initialization process being terminated by a power off before all of the components of the zero or initialization operations were completed. This resulted in not all counts and data being correctly cleared.

Sequoia has added an indicator bit, which resides in the battery backed-up static RAM in the Memory Pack. The bit is set before the zero or initialization process starts and is not cleared until the process has successfully completed. If the power is removed, or the operation interrupted, the process will start again, at the beginning, the next time that the system is restarted. This will continue until the initialization or zero process has completed.

Conclusion:

MITIGATED Based on the static code review performed on both the version used for the State of Washington testing and the current version, this new mechanism that verifies successful completion of the initialize or zero operation should prevent occurrence of the noted error.

5.6 Assessment of new components (WinEDS Extended Services and WinEDS Election Reporting) vulnerabilities

5.6.1 Vulnerabilities related to code and data integrity

Finding:

The new components “WinEDS Extended services” and “WinEDS Election Reporting” are susceptible to SQL injection attacks from at least two vectors[*59]:

1. via unfiltered input from a GUI
2. via specially created (malicious) input files

The software fails to detect and/or mitigate malicious user input.

Implication:

Direct access to database functions and structures is possible without major effort. A malicious user might cause data corruption and/or incomplete or false results. The system might enter an insecure state.

5.6.2 Design Issues

Finding:

The system explicitly relies on the user to ensure data integrity[*60]. Rather than implementing a respective safeguard itself, the system instructs the user to perform the necessary step.

Implication:

By not acting as instructed, a user might cause data corruption and/or incomplete or false results. The system might enter an insecure state.

5.6.3 Programming Style

Finding:

In several modules, clear, structured error handling is missing[*61]. The programming languages used would easily allow applying structured error handling – this has not been done. In at least one case, the error was signaled ("in band") as a string with a specific content – the correct result would have been a hash string consisting of hexadecimal characters[*62]. In several other cases, the fact that routines could return errors has simply been ignored. In at least one example, the comments do not agree with the code[*63].

Implication:

Calling modules do not know about error conditions that may cause data corruption and/or incomplete or false results. The system might enter an insecure state. In the case of disagreement between code and comments, the issue does not have any direct implications on function; however, it makes maintenance of the code more prone to errors.

5.7 Assessment of Integrity Protection of Ballot Data and Images for the 400-C Central Count Scanner and Optech Insight® Plus

5.7.1 400-C Central Count Scanner

The 400-C Central Count Scanner ultimately fails to provide reasonable integrity protection of election information crucial for the success of an election. The subsystem employs an unverified proprietary implementation of the SHA-1 cryptographic hash and CRC checksum algorithm in order to check the integrity of ballot and election data loaded from a removable media disk. The hash digests of the files that are created are stored in the files themselves, and since there is no secret information used in this hash, an attacker can intercept, modify, and recompute the hash without significant effort.

When election information is loaded from a network connection rather than a removable media disk, the hashes are not computed. This, along with the fact there is no evidence to show the network connection is authenticated and integrity checked through SSL or TLS, creates a plausible opportunity for an attacker to affect election information without detection.

Furthermore, certain portions of the software only use the CRC algorithm to integrity check the precinct files. Using a CRC checksum is adequate for random error checking, but will not prevent malicious attacks on the integrity of the data.

5.7.2 Optech Insight® Plus

5.7.2.1 Vulnerabilities related to code and data integrity

Except for a simple CRC done on the Memory Pack to check integrity (this test is only useful from a data damage standpoint, not security as the CRC can be trivially recalculated), there is no security on the data in the Memory Pack at all, it is just a bulk data handling device (like a plain USB drive before USB), with some support for particular data structures (precinct data, code pages, etc). Program code or data could be easily manipulated by an attacker using an EEPROM/FLASH burner to rewrite the MPR or Insight Plus® APX

programming. An EEPROM/FLASH burner or an MPR could be used to rewrite the contents of the data or APX programming in the MemoryPack.

5.7.2.2 Passwords (Identification and authentication)

There is no authentication or access control to protect the contents of the Memory Pack

5.7.2.3 Encryption (Cryptographic algorithms and key management)

None used

5.7.2.4 Audit reliability (Audit log management)

See section above; there is no significant protection for any of the data

5.8 Analysis of cryptographic algorithm implementations

During the review of the Sequoia source code, the atsec team found an implementation of the Advanced Encryption Standard (AES) in the WinEDS subsystem. In order to make sure that this code base and the future extensions of it could rely on a sound cryptographic library, the reviewers put the candidate implementation through formal testing by atsec's Cryptographic Security Testing laboratory, which utilizes the NIST-developed Cryptographic Algorithm and Validation System (CAVS) tool.

The reviewers also found implementations of the SHA algorithm that could be tested by this method, but they were unable to complete formal testing in the time available.

The Computer Security Division at the U.S. National Institute of Standards and Technology (NIST) maintains a number of cryptographic standards, and coordinates algorithm validation test suites for many of those standards. The Cryptographic Algorithm Validation Program (CAVP) encompasses validation testing for FIPS-approved and NIST-recommended cryptographic algorithms. This list includes many popular algorithms, such as the Advanced Encryption Standard (AES), Triple DES (TDES), Skipjack and Digital Signature Standard (DSS) (DSA, RSA, and ECDSA) algorithms.

The CAVP certifies that these algorithms are implemented correctly through formal testing supervised by accredited testing laboratories who generate the test vectors that can verify the correctness of the algorithm implementation in accordance with the test described in the associated validation system document.

The certification program was instigated to provide assurance that cryptographic algorithms are implemented correctly in cryptographic modules. CAVP certification is mandatory for all modules being certified as conformant with the FIPS 140-2 standard. NIST statistics indicate that close to 25% of algorithms tested showed errors in implementation that were corrected as a result of the testing process. CAVS testing of algorithms is a pre-requisite to FIPS 140-2 testing that is specified in the Voting System Standards 2002.

It should be pointed out that CAVP certification does not by itself provide any assurance that the algorithm itself is sound. It does, however, provide assurance that the algorithm was implemented correctly.

The AES algorithm implementation passed the testing in the laboratory and awaits formal validation of the result by NIST's Cryptographic Algorithm Validation Program. Therefore, we feel that this implementation is sound and, when used correctly, will provide the software with the cryptographic functionality provided by the AES standard.

5.9 Assessment of overall system security posture and potential attack scenarios

After performing a static code review of the Sequoia Voting System v4.0 source code, the reviewers found numerous security vulnerabilities and confirmed that many of the issues described in the UC Berkeley report have not been addressed. The reviewers found that the vulnerability found during certification testing by the State of Washington has been successfully resolved.

Some issues in the UC Berkley report have been mitigated and contribute directly to an improvement in the overall security posture, while several others have been addressed and will form the basis of an improvement of the security posture after other issues are addressed. At this time, however, the reviewers note that there is enough evidence to demonstrate that the integrity of election definitions and ballot information is still not properly protected. Many of the attack scenarios identified center around interception and modification (integrity) of data, since there are no reliable ways implemented to detect them. The following is a non-exhaustive list of potential attack scenarios.

5.9.1 Attacking the SQL database directly

Anyone who has access to the network that the WinEDS SQL Server is a part of and any account on the WinEDS system (regardless of the assigned permissions) could bypass all security measures enforced by the WinEDS Workstation by connecting directly to the SQL server. This is possible because for every user created through the WinEDS workstation, a user with administrative privileges is created in the SQL database. Accessing the SQL server directly using publicly-available client software could, therefore, allow the attacker to arbitrarily affect the results of the election.

5.9.2 Attacking the WinEDS subsystem

Anyone who has access to the network that the WinEDS SQL Server and the WinEDS Workstation are a part of, can conceivably monitor and affect communications exchanged between the two subsystems. Because the network connections are not encrypted, authenticated, or integrity-checked (through, for example, SSL or TLS), changes to the network communication stream will not be easily detected. This provides an avenue for arbitrary SQL statements to be executed on the WinEDS SQL Server, thereby providing the attacker complete control over the election results.

5.9.3 Attacking the transfer of data between the 400-C and the WinEDS

Anyone with access to the removable media (for example, a USB memory stick or a floppy disk) that carries election information between the 400-C and the WinEDS subsystem could manipulate election results. Since the hardware for transferring the election information is readily available to the public, it would be fairly easy for an attacker to falsify records. In fact, the attacker has no need for direct access to the legitimate media, but simply needs to replace that media with one that contains tampered results.

5.9.4 Attacking the transfer of data between Optech Insight and the MPR

Anyone who gains access to a MemoryPack that contains election information and has the means to edit that information could falsify results by simply editing the data, recomputing the CRC checksum, and writing it back to the MemoryPack.

5.9.5 Attacking the Optech Insight® directly

Poll workers, election officials, or storage personnel having access to the Optech Insight®, a MemoryPack, and the means to write to that MemoryPack (an MPR or an EEPROM/FLASH burner), could install and

execute arbitrary software on the Optech Insight®. Using this altered MemoryPack, it is possible to manipulate election results without being present when the results are being stored on the MemoryPack or later when they are tabulated.

6 Conclusions

The source code provided as evidence was examined by a team of atsec information security corporation consultants. The focus was on determining whether the provided source code resolves issues discovered during earlier testing, which were identified in the UC Berkeley report (as both high-level security architecture issues and specific security defects) and in the Theisen testimony [THEISEN]; top-to-bottom review of two new modules (WinEDS Extended Services and WinEDS Election Reporting); and evaluating the extent to which the system protects the integrity of ballot data or ballot images stored on the 400-C Central Count Scanner and in the Optech Insight® Plus.

With regard to determining whether the provided source code resolves the high-level security architectural issues identified in the UC Berkeley report, the reviewer found that the previously reported security architecture issues remain issues in the current version. There still is no effective mechanism to protect the integrity of data that is transferred between components of the system via removable media; there is a potential vulnerability for SQL injection attacks that result in unauthorized access to election data stored in the database or execution of malicious code on the database server machine to crash the system; a user can exploit a system weakness that enables him or her to access the database without going through the WinEDS user interface, and once there, that user can add, delete and modify any data in the database; cryptographic methods are improperly used; access control management is still cumbersome and subject to user error and also can be circumvented; and while password management has been improved, because of an architecture defect, the strengthening of password management does not necessarily lead to a strengthened access control system.

With regard to determining whether the provided source code resolves specific security defects identified in the UC Berkeley report, the reviewer could verify that nine of the 47 previously reported defects have been sufficiently resolved in the provided source code to mitigate the identified vulnerability. Code modifications for two defects partially resolve the reported issues. Code modifications for two defects do not sufficiently mitigate the reported vulnerabilities they are intended to resolve. Resolution of 10 issues could not be determined by static review of the source code, but can only be verified by functional testing, penetrating testing, live-code debugging, or other means; the reviewer acknowledges that two of these issues might be resolved. Based on the code review, the reviewer found that approximately 24 of the 47 issues have not been addressed by code modifications.

With regard to determining whether the provided source code resolves specific defects identified in the Theisen testimony, the reviewer found that a new mechanism that verifies successful completion of the initialize or zero operation should prevent occurrence of the previously identified error.

With regard to review of the two new modules (WinEDS Extended Services and WinEDS Election Reporting) the reviewer found that the modules are susceptible to SQL injection attacks via both the GUI and malicious input files; rely on user action to ensure data integrity rather than implementing a system safeguard; and provide inadequate error handling. Exploitation of any of these weaknesses could result in data corruption and/or incomplete or false results. The system could enter an insecure state.

With regard to evaluating the extent to which the system protects the integrity of ballot data or ballot images stored in the 400-C Central Count Scanner and Optech Insight® Plus, the reviewer found that except for a simple CRC check, there is no security on the data in the MemoryPack. Program code or data could be easily manipulated by an attacker.

Overall, the reviewer found that while progress has been made, the integrity of election definitions and ballot information is not properly protected. Many attack scenarios center around interception and modification of data, since there are no reliable ways to detect them.

7 References

[VSS]	2002 Voluntary Voting System Standards, Volume I, sections 4 and 7 and Volume II, section 5
[FIPS197]	FIPS 197
[FIPS180]	FIPS 180-2 with Change Notice 1 (February 25, 2004)
[SOW]	atsec statement of work, 2008-08-15
[UCB]	Source Code Review of the Sequoia Voting System, University of California Berkeley Report, July 20, 2007
[THEISEN]	TheisenTestimony, Ellen Theisen (VotersUnite!), May 20, 2008
[PIERCE]	Pierce Error Correction, 2008-09-12
[WinEDS_ INSTALL]	Sequoia Voting Systems WinEDS Installation Guide, Release 4.0, Document Version 1.04, July 2008
[WinEDS_ SYSOP]	Sequoia Voting Systems WinEDS System Operations Procedures, Release 4.0, Document Version 1.05, July 2008
[WinEDS_ERO]	WinEDS Election Reporting Operator's Guide, Release 4.0, Document Version 2.05, June 2008

8 Trademarks

Microsoft and SQL Server are registered trademarks of Microsoft Corporation in the United States, other countries, or both

Sequoia Voting Systems and Optech Insight are registered trademarks of Sequoia Voting Systems